Production-Level Distributed Parametric Study Capabilities for the Grid

Maurice Yarrow[1], Karen M. McCann[1], Edward Tejnil[2], and Adrian DeVivo[1]

[1] Computer Sciences Corporation, Mail Stop T27A-1, NASA Ames Research Center, Moffett Field, CA 94035, USA
{yarrow,mccann,devivo}@nas.nasa.gov
[2] Eloret Institute, Mail Stop T27B-1, NASA Ames Research Center, Moffett Field, CA 94035, USA
tejnil@nas.nasa.gov

NASA Ames Research Center
NAS Technical Report NAS-01-009
October 2001

Abstract. Though tools are available for creating and launching parameter studies in distributed environments, production-level users have shunned these tools for a variety of reasons. Ultimately, this is simply a result of the inability of these tools to provide anything more than a demonstration-level capability, rather than the flexibility and variety of industrial-strength capabilities that users actually require. In addition, despite the difficulties of creating parametric studies without specialized tools, users still demand that such tools be intuitive, easy to use, and versatile enough to support their particular experimental procedures. We show some solutions to real problems encountered in users' parametric experiments, and simultaneously show how the success of grid computing in general will rely on the ability of grid tool developers to provide a much greater level of capability and generality than users have seen in current grid-tool demonstrations.

1 Motivation and Background

The ILab parameter study creation and job submission tool [1, 2] was developed to fulfill several goals. First, it had to be grid-enabled. Currently, ILab can launch parameter studies onto remote resources either with or without metacomputing middleware such as Globus [3]. Second, ILab had to be easy and intuitive to use. To accomplish this we constructed a sophisticated graphical user interface (GUI). We equipped it with current user interface paradigms and conveniences, including a special purpose parameterizing editor, file browsers, "previous-and-next" dialog widgets, built-in help for all screens, a graphical data-flow-diagram mechanism for constructing complex processes, etc. ILab had to be modular and easily extensible, so we developed ILab in object-oriented Perl (an excellent language for rapid-prototyping) with Perl/Tk for the GUI components. We also developed a separate GUI generator to facilitate the rapid construction of new GUI components. For extensibility, we designed ILab as basically a front-end which collects the many specifics of a user's experiment, and a back-end, consisting of five different "job models" (described below), which are modular code generators that produce Korn shell scripts. Currently, these job models build the scripts which constitute the set of individual jobs comprising a user's parameter study experiment. Adding a new job model is relatively straightforward.

The five current job models are (1) a Local job model for an experiment which will run entirely on the local machine on which the user is running ILab; (2) a Remote job model, where each job in the experiment consists of a script that runs locally and chooses a remote machine from a list of candidate machines to which it will migrate a second script which actually runs the user job on the target machine, either under control of a job scheduler such as PBS or without; (3) a Globus job model similar to the Remote job model, but leveraging the Globus toolkit of command-line functions; (4) a Restart job model, which we will describe in detail below; and (5) a Condor job model, under development.

Having developed a powerful tool for parameter study creation and launching, we endeavored to attract real users from the NASA community of aeronautics engineers and researchers. Users were impressed to find that ILab, ostensibly a research tool written for experimenting with grid middleware, distributed computation models, and user-interface issues, was more sophisticated and capable than they had expected. Nevertheless, they could not use it for their highly specialized parameter studies for a variety of reasons. During our design phase for ILab, we specifically solicited user input from our potential user community. Users initially described a usage scenario with simple common needs. We naively assumed that users would be able or willing to accept a simplified view of their parameter study experiments. When we presented the developed product to these users, almost all of them nevertheless stubbornly decided to continue to use the specialized shell or Perl scripts that they had developed or had had developed for their needs. These scripts invariably were so specific that they lacked generality in terms of what engineering codes they could run, the layout of the file systems that they used, the parameterization of the input files for running, etc. Thus, these users' scripts were not usable for any problems except their own. On the other hand, the models for parameter studies experiments which ILab supported, sensible as they were, failed to address functionality critical to individual user environments. We will give just a few examples.

Some users required a large number of PBS options not supported by Globus. For example, many PBS users require PBS flags which allow them to get mail notification of job completion depending on completion status, to designate specific nodes on a given machine which should be used for their computations, or to specify a time after which a job is eligible for execution. Thus, it must be possible to specify any number of arbitrary PBS options, including any number of resources in a PBS resource list.

Some users needed to be able to rename groups of files with a common prefix or a common suffix. That is, a program run might have to rename all files of the form q.1.1000, q.2.1000, etc, to be of the form q.1.restart, q.2.restart, etc. This is an operation for which no particular shell command capability exists. It is a fairly specialized rename operation, and not easily accomplished, especially for the general case of a rename of all files of form1 to be in form2, where form1 will contain wild cards, and form2 may need to contain wild cards.

Some users needed to be able to obtain input files and executables from a variety of locations on arbitrary remote machines. Subsequent to completion of computations, users had to be able to send output to arbitrary archive facilities. Since such facilities are not within the Globus testbed, this posed a problem. Many users needed to be able to specify that the individual jobs comprising their parameter study should run in a particular scratch directory partition. However, the name of this partition could vary from machine to machine. For example, on machine1 it might be /scratch/username but on machine2 it might be /SCR/username.

Though some of the specialized needs of users might be addressable by skillful shell programming on their part, ILab makes the assumption that a user has no scripting knowledge. However, ILab does permit the entry of arbitrary shell commands, but does not assume that users wish to address specialized functional needs via the shell.

Another important and challenging issue was a "restart" capability, which many potential users required. (This is not to be confused with a checkpointing capability.) Users need this capability for two primary reasons. First, real production environments require that jobs be submitted through a scheduler, but many users have job runs that require hundreds or thousand of hours of computation. Such jobs will typically be of low priority and will not be given a run state by the scheduler for days in real environments. Users therefore wrote PBS scheduler scripts which would request less compute time but would resubmit themselves to the queue upon completion. In this manner, jobs could be configured to fit in a queue with relatively high priority and the entire run could be expedited.

There is a second and related issue. For many of the fluid dynamics computations common within NASA, various solver parameters (not physical parameters) need to be adjusted or "ramped" during the computation, and many of the legacy flow solvers did not provide a mechanism for specifying a schedule that automatically modified these solver parameters during computation. To accommodate this, it should be possible for solver parameters to be changed in conjunction with job restarts. This amounts to a variety of computational steering.

Because of the need for these functionalities, users rejected ILab in favor of the more cumbersome and specialized scripts developed for their parameter studies. As a result, we decided to add many of these user-requested features.

2 Addressing the User Need for a Restart Capability

The simpler specialized needs of users, such as generalized bulk file renaming, file archiving issues, etc., were addressed largely as ILab user-interface issues. Solutions for some of these were straightforward options added to ILab in such a way as not to violate our rule of "no user programming required". The generalized bulk file renaming, for example, required both a graphical-user-interface option, and underlying support in the form of a complex Korn shell function added to the shell scripts that ILab generates. By far the most complex capability added was the "restart" option. This required added extensive GUI capability, considerable additional internal data structure support within ILab, and a completely new job model for shell script generation, including a server script for launching the restarts. We will explain in detail how this capability was implemented with respect to this new job model, the nature of communication occurring between the generated shell scripts, and the server required to perform these restarts.

The RestartJobModel, like both the RemoteJobModel and the GlobusJobModel, generates Korn shell scripts. Currently, inter-machine functionality within these scripts is implemented using the secure shell "ssh" command and the secure copy "scp" program (or "rsh" and "rcp", if the user requests these). This job model is not implemented using

Globus because too few production and archive machines of users' choice are members of the Globus testbed, and so-called "third-party delegation" would be required on all remote systems for implementing the restart functionality.

For each separate job in the user's parameter study, two shell scripts are generated, one being a script which runs locally (i.e., on the originating machine), and the other being a script which is migrated to, and runs on, the remote system that is chosen for computation by the local script. The local script performs the following functions. First, it queries remote systems from the list of candidate machines that have been selected by the user. Via ssh, the local script then interrogates the PBS queue on each system in turn, until it finds a suitable system onto which it can submit the job with hope that the job will start immediately. If all systems have queued jobs belonging to the user, the local script will go into a polling loop, and periodically repeat the queue interrogation for the system list until a suitable candidate machine is found. When such a system is selected, the local script, again via ssh, first creates an appropriate directory and subdirectory on the remote file system partition indicated by the user, and then, via scp, migrates the second shell script to that subdirectory. The local script then submits the migrated script (which includes whatever PBS directives are required) to the remote PBS queue, and terminates.

This is similar to the sequence of operations that occur in the RemoteJobModel and in the GlobusJobModel, with the following important difference. The very first of the local scripts to run launches a server script which will run for the duration of the parameter study (see Fig. 1). It is the function of this server to poll for the presence of semaphore files sent to the originating machine by each of the parameter study jobs upon their completion. This is how the server will know which job and which phase to restart.

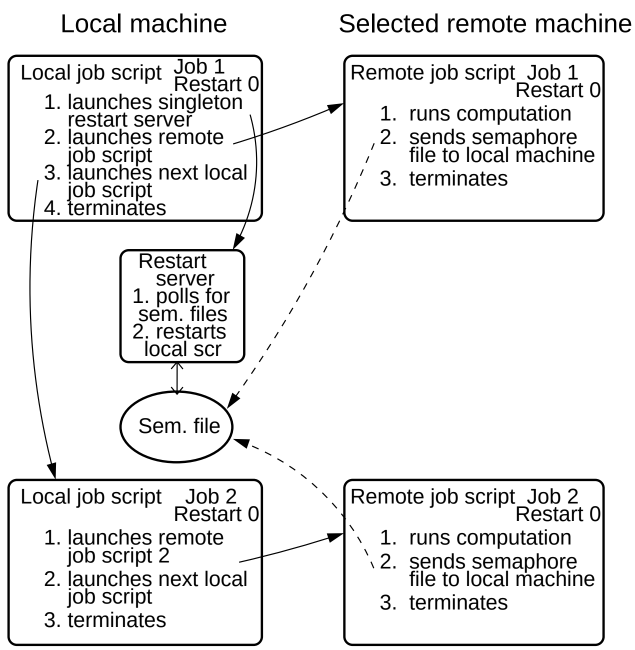

Fig. 1. Restart job model scripts, for restart 0

When PBS runs the remote script, the following sequence of operations occurs. First, files required as input for the compute executables are obtained from their source locations using cp if local or scp if non-local. Next, the parameterized input files are copied from the originating machine, in particular, the input file associated with the first phase (restart "0", as it were). The compute executables are run, each preceded by the staging of appropriate input files. Upon completion of computation, all files, both input and output, are archived to a user-designated archive system and are placed into a directory structure (created by the remote script), which is comparable to the directory structure in which the remote script itself ran. An additional archive subdirectory structure is created to accommodate the distinction between files required by, but not changed by, the computations, and files changed or created by the computations (output files). The final act of the remote script is to send a semaphore file to the originating machine which indicates which restart stage of which job in the parameter study experiment has just completed. It is this semaphore file from which the restart server will determine the correct local script to re-run (see Fig. 2). This appropriate local script then repeats its search for the best-candidate remote compute machine, and resubmits the remote script to that machine, this time for the next phase of the computations (restart 1 through final restart) of this particular job in the experiment. When this job restarts, it knows its own restart number, and thus knows from which archive directory to obtain the appropriate input files so that computation can be resumed.

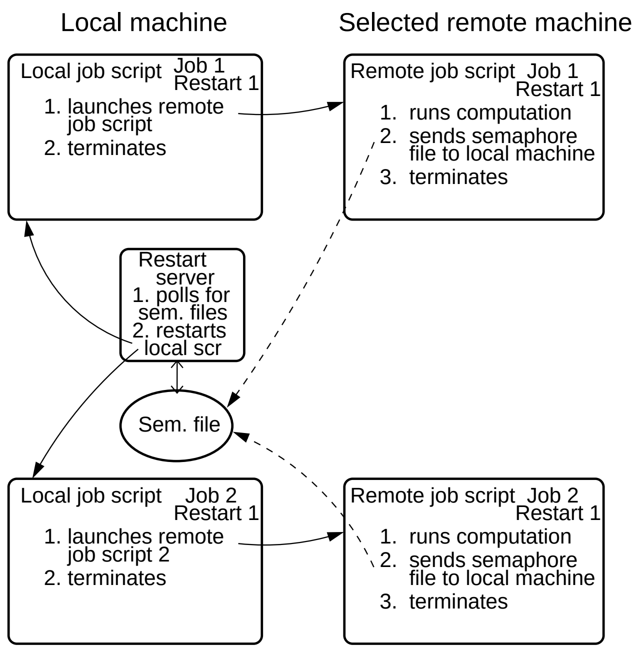

Fig. 2. Restart job model scripts, for restart 1

A special case of this processing is when the user requests of this job model that no restarts be performed (restart "0" only). In this case, only one stage of computation is performed, followed by archiving of all output files, making the RestartJobModel perform like the simpler RemoteJobModel.

3 Lessons Learned from the Restart Job Model Implementation

At first glance, it would seem that the restart capability is a very specialized mode for distributed parameter study usage. Indeed, only a subset of all users wishing to run parameter studies will need the restart capability, though at NASA's Ames Research Center, that subset is surprisingly large. Discussions with production users revealed that very few researchers and engineers who wished to generate parameter studies were content with simple models for issues such as archiving, schedulers, etc. Our conclusion is that real users have complex and varied needs which made almost every aspect of their interactions with production-level systems non-addressable by demonstration level tools. Grid-enabled tools for distributed tasks most often make use of simplifying models for conducting their business. These have limited practical utility. We have found that designing usable models which do not require any programming expertise from users is a considerable challenge. The success that tool designers and architects have in meeting this challenge is going to have a significant impact on the long-term success of computational grids and the underlying middleware.

4 Some Additional Production-Level Issues

We list here several additional issues that have been encountered in our production computing environments. We are considering or implementing solutions to these issues at this time.

Copying of files from source system to target compute system, and from compute system to archive system, is accomplished with the Unix secure copy command "scp" and, for the GlobusJobModel, with the "gsi" enabled version of scp. We have seen the scp command stall when archiving files. We are considering accompanying each scp command with an additional background process to monitor the progress of file copying. Then, if the scp process stalls for some period of time, the monitor process will terminate the copy. An additional attempt at copying could be made, or this entire particular job in the Experiment can be declared non-viable and aborted.

Many supercomputer systems are actually accessible only through a "front-end" system. Currently, some aspects of our job models require the ability to invoke the "ssh" command directly onto the compute system in order to perform resource discovery or to submit a job onto a batch scheduling system. On our local systems at the NAS division at Ames Research Center we have been able to issue ssh commands directly onto compute engines only because the duration of these commands is brief. Were these commands to last any significant time, they would be terminated by "skulker" programs monitoring command traffic to these systems. We will have to build into ILab a new layer of resource configuration data and control in order to accommodate indirect front-end access to arbitrary compute server systems.

ILab is currently incapable of automatically terminating all remote jobs in an Experiment. Since this could be very costly to a user in the case of an incorrectly configured Experiment, we will be adding the ability to terminate all Experiment jobs and clean up all appropriate disk systems.

These are only a few examples of problems that must be addressed when computing on the grid.

5 ILAB's CAD (Computer Assisted Design) screen: Directed Graph

Fig. 3 displays a Directed Graph whose nodes are units of user's ILab Experiment. The current icon units are Experiment, Process, FileList, FileHandling, and Restart; we anticipate adding additional icons for Loop, Condition, and Branch.

For each Experiment data set, there is one Experiment icon, representing global data for the Experiment: systems for running the Experiment, remote root and archive directories, etc. For each Process in the Experiment, there is one Process icon, which represents data specific to a Process: original path, meta-environments used, command line arguments, etc. If a Process has any associated files, a FileList icon is placed beneath the Process icon; the associated data is a list of file paths and names necessary for that process. If there is to be any handling (renaming, archiving, moving, etc.) of any of the files in the Process File List, a FileHandling icon is placed below the Process icon, to the right of the FileList icon. If user wishes to restart the Experiment, a Restart icon is placed beneath all Process icons, and connected to the first and last Process icons. The associated Restart data specifies number of restarts, data to be changed for each restart, restart stop conditions, etc.

Note that a restart is not the same as a loop; a loop would mean repeating the execution of one or more processes, while a restart means the repetition of all Experiment processes.

The set of icons representing Experiment data is interconnected by arrows which signify the flow of execution within the Experiment, and also by straight lines which signify attached data and/or conditions.

The CAD screen has a dual functionality. First, for Experiment Editing, the directed graph is generated from pre-existing Experiment data when user chooses "CAD Screen" from the "Edit Experiment" pull-down menubutton. User can pop-up editing dialogs from each of the icons, or can add and remove icons as necessary. Second, for Experiment creation, a user can initialize an Experiment, and then add icons and enter data for each icon. (Icon display is color-coded to signify whether necessary data has been entered.) As each icon is added, the corresponding Experiment data structures are generated; when all necessary data has been entered into these structures, user can save the Experiment data to disk, and/or execute the Experiment. The CAD screen also functions as an organizational aid in Experiment execution and tracking; it is easier to visualize the parts of the Experiment and the flow of execution from the CAD screen, than from the wizard-dialog-based entry of the same data.

The form of the CAD screen layout was heavily influenced by practical considerations, since there are two non-trivial problems to overcome in order to create a visually effective display. The first problem is the placement of icons in such a way that no icons overlap. We solved this problem in two ways: by placing icons automatically in a simple 2-D "grid" type layout, and by providing, as a backup where the first method might fail, user ability to move icons, and their attached lines and arrows. In this way, if any icons overlap, user can easily and quickly adjust the icon positions in order to get a "neat" display.

The second problem we had to face was the likely proliferation of icons for real-world Experiment layouts: we wanted to avoid the situation where the directed graph would get so large that only a small portion of it could be displayed at once on a computer screen. Our approach was to make the icons represent "high level" data and operations, as opposed to "atomic" data and operations: each icon accurately reflects the internal structure of the major ILab data

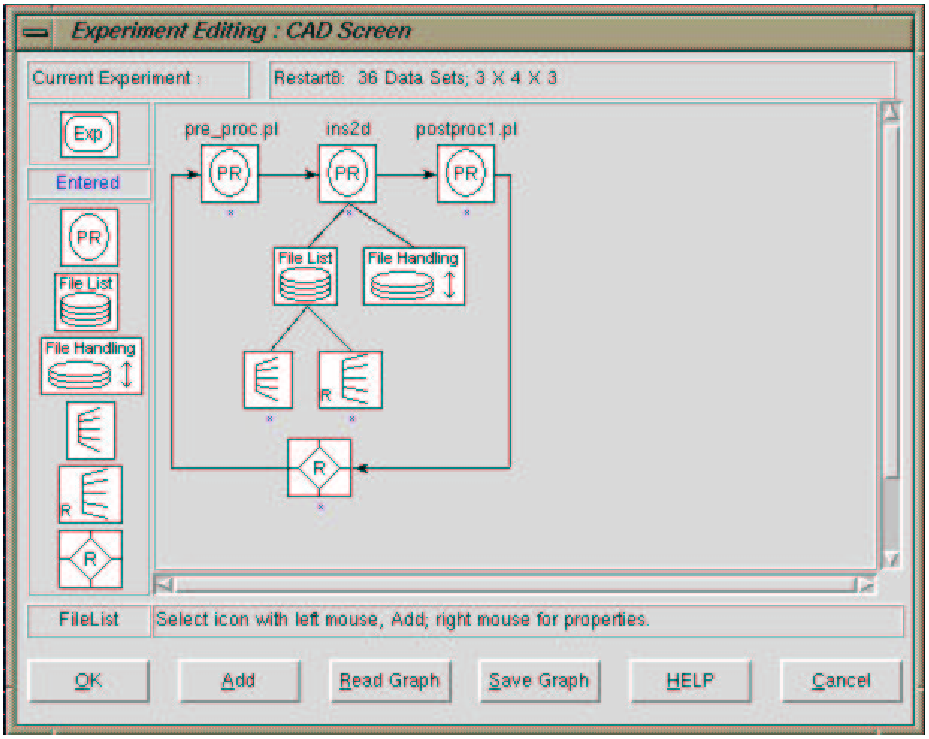

Fig. 3. CAD process specification screen

objects. This approach is both intuitive and effective, since a very large graph might provide more detail, but would fail in the effort to give user a reasonably effective visual cue to the organization of an ILab experiment.

6 Future Development of the CAD Screen

The inclusion of branching and loops is planned for the next release of ILab. Branching can be caused by multiple levels of parameterization, or by repeating the output of one process to several different subsequent processes. Conditional loops are repeated executions of one or more Experiment processes, where the repetition is terminated by either file existence, a process return value, or the execution of a predetermined number of repetitions. All of these conditions represent non-linear paths of process execution. It is anticipated that ILab's script generation algorithms will have to be significantly extended, perhaps by generating layers and/or sets of execution scripts, in order to handle these cases. We also plan to include the directed graph display in ILab's Logbook pages, for user convenience in referencing the Experiment results, and the construction of reports.

7 ILab's Interface to Condor: Under Development

We are collaborating with A. Globus (NASA Ames Research Center, Computer Sciences Corporation) to add ILab options for the Condor High-Throughput Computing System [4]. This aspect of ILab development is currently of interest, since the Condor pool at the NAS systems division at Ames Research Center now contains approximately 350 systems, and more systems are being added. We anticipate that ILab-generated Condor experiments will be of use to the scientific community, especially in the area of genetic algorithms, which need many runs to generate results. A user who already has Condor installed on his/her system will be able to use ILab to create large sets of program runs and automatically gather the output back to the originating system for processing. ILab will generate directories, appropriate lists of input files, and multiple Condor submission scripts. For significantly large runs, this assistance will be non-trivial, since creating file lists and Condor scripts "by hand" is quite onerous, time-consuming, and error-prone.

The Condor system has built-in monitoring and restart capability, but currently lacks a simple way of executing more than one process in sequence. If several separate programs are submitted to Condor, Condor guarantees that each

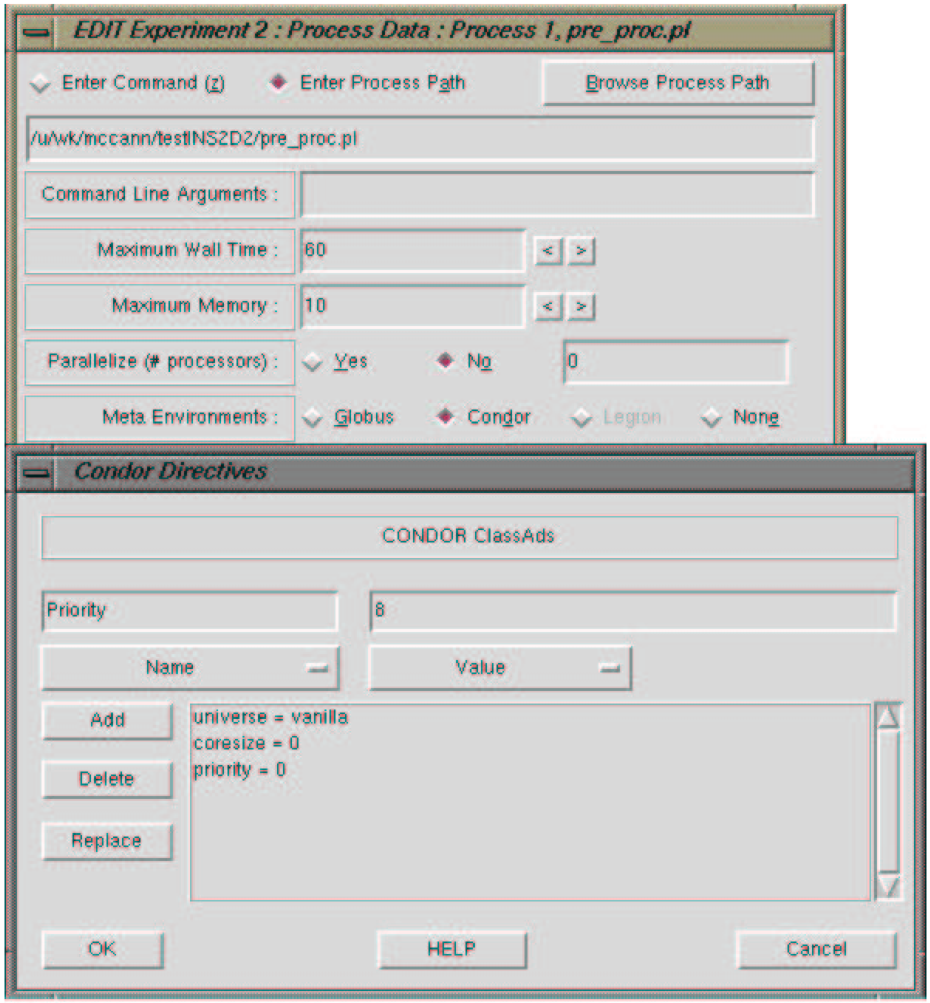

Fig. 4. Condor job model screen

program will be executed, but does not guarantee the order of execution, since Condor implements run-once-and-only-once semantics. In ILab, we generate scripts which execute all processes in user's Experiment in sequence, and each script is itself submitted to Condor as a Condor process, by a separately generated Condor "submit" script. In order to execute the set of scripts, the Condor "universe" must be specified as "vanilla", meaning that a script cannot be restarted from an interim point in case of failure, but must be re-executed from its beginning. This means that ILab's "Restart" option is not usable in Condor experiments. However, this should not be an issue for Condor jobs, since the restart option has been designed for use on multi-processor supercomputers with queue time limits, and for the specific case of Computational Fluid Dynamics (CFD) flow solvers that need to ramp-up certain internal parameters. We will also investigate the new Condor extension "DAGMan", which refers to a directed graph sequence of processes, as an alternate solution to this problem.

Setting up a Condor experiment in ILab is very simple, and involves two steps. First, users must choose the Condor "MetaEnvironment" in the Process specification dialog (see Fig. 4). Second, from the the Condor Options pop-up dialog, user must enter Condor "ClassAds" specifying execution environments, memory requirements, core size, etc. Some necessary default ClassAds are supplied by ILab. Since the number of ClassAds is rather large, and their construction fairly complex (users may specify logical "and" and logical "or" within each ClassAd), at this time we have not added extensive error-checking and user aids for the ClassAds. Instead, ILab presumes that a Condor user already has background knowledge regarding the construction of Condor submission scripts, and users are referred to the Condor web site for information. Nevertheless, entry of ClassAds is still very easy, and some ClassAd names are available from pull-down menus for user's convenience. After choosing Condor and entering necessary ClassAds, users will proceed as with other ILab experiments: the generation of parameterized files, and the generation and monitoring of scripts, will operate as for other MetaEnvironments. We plan to further implement the Condor option in a few months, and we will provide documentation that includes sample ClassAds, and some basic instructions, for those users who are unfamiliar with Condor.

8 Conclusions

We have labored much over details here largely because production-level grid computing cannot be accomplished without strenuous attention to details. Users justifiably demand services well beyond the demonstration level. We have described some typical user requirements and corresponding solutions which involved considerable development effort. We believe that the "Grid" is now at a critical juncture where this level of effort is necessary. It will be the production-level users whose acceptance will determine the success of the Grid.

References

1. Yarrow, M., McCann, K. M., Biswas, R., Van der Wijngaart, R.: An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid. Grid Computing - GRID 2000, First IEEE/ACM International Workshop on Grid Computing, Bangalore, India, December 17, 2000, Proceedings.
2. DeVivo, A., Yarrow, M., McCann, K. M.: A Comparison of Parameter Study Creation and Job Submission Tools. Available at http://www.nas.nasa.gov/Research/Reports/Techreports/2001/nas-01-002-abstract.html.
3. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 11(2):115-128, 1997.
4. Litzkow, M., Livny, M.: Experience With The Condor Distributed Batch System. IEEE Workshop on Experimental Distributed Systems, Oct. 1990, Huntsville, Al. Available at http://www.cs.wisc.edu/condor/publications.html.